EXHIBIT 5

STARKs on Bitcoin: From Outside In

Laolu Osuntokun
STARKWARE Sessions 2023
@roasbeef

Feb 2nd, 2023

Overview

- Why STARKs on Bitcoin?
- Practical Multi-Pronged Ecosystem Integration
- STARKs on Bitcoin p2p Network
- Taro
- zk-Taro
- Future Directions

Why STARKs on Bitcoin?

- Privacy
  - General method to improve on-chain privacy within the system
  - Tools like Confidential Transaction are cool, but only hide the input+output amounts
  - Richer succinct zero knowledge proofs can compress entire histories (transaction graph) into a single transaction
- Scalability
  - Ability to pack in more transactions per-byte
    - Fundamental motivation of the Lightning Network!
  - Logarithmic scaling -> batching++; efficiency++; utilization++;
  - Most Bitcoin transactions these days larger than you think
    - 65 KB consolidation transaction (200 inputs)
- Transparency
  - Bitcoin ethos fundamentally opposed to “trusted set ups” - ~~CRS~~ 😈
  - Conservative DNA of Bitcoin cryptographic design -> “moon math” unappetizing
  - Hash functions + merkle trees + polynomials 😇

Practical Multi-Pronged Ecosystem Integration

- Ignoring soft-fork coordination costs:
  - Is the stack ready for primetime on the #1 blockchain in the world?
  - Not quite yet.... 😅
- How can we de-risk, gain mindshare, build excitement, and practical experience?
  - Go from the outside in!
  - Start on the higher layers (LN, Taro), p2p network, Bitcoin-native applications
  - Build up maturity of the periphery of the system before rolling up sleeves for the consensus layer
  - Learnings related to UTXO-oriented STARK design to shape eventual soft fork proposal

STARKs on Bitcoin: Peer-to-Peer Sync - Light Clients

- Leverage succinct computation integrity to allow for “instant” sync times
  - Amazing project tackling this design space: ZeroSync
- Light clients (neutrino, etc):
  - Skip fetching from p2p network entirely (rather remove p2p interaction)
  - STARK sketch:
    - Generate incremental/recursive proof for header connectedness + validity starting from genesis to chain tip
      - f(header_{n-1}, header{n}) -> header{n}
        - PoW check, double-sha, retargeting, etc, etc
    - Client downloads blob of all 750k headers (80 byte headers, 60 MB total)
      - Alternatively just final header+genesis
    - Download+verify STARK proof, join p2p network, gg
  - Other applications:
    - LN Channel Graph bootstrap
    - BIP 158 filter verification

STARKs on Bitcoin: Peer-to-Peer Sync - Full Nodes

- Full nodes (btcd, bcoin, libbitcoin, bitcoind, etc):
  - Beyond just header state, also want to prove UTXO state transition validity
    - f(chainState, newBlock) -> newChainState
  - utreexo
    - Forest of complete merkle trees representing UTXO state
    - During proving, rather than look up into entire UTXO state (~5 GB) prove inclusion of inputs
  - Given utreexo root + final state, able to start to validate incoming blocks
  - Ultimately requires full Bitcoin Script VM implementation in Cairo!
    - Potential for speed ups via optimization and built-ins

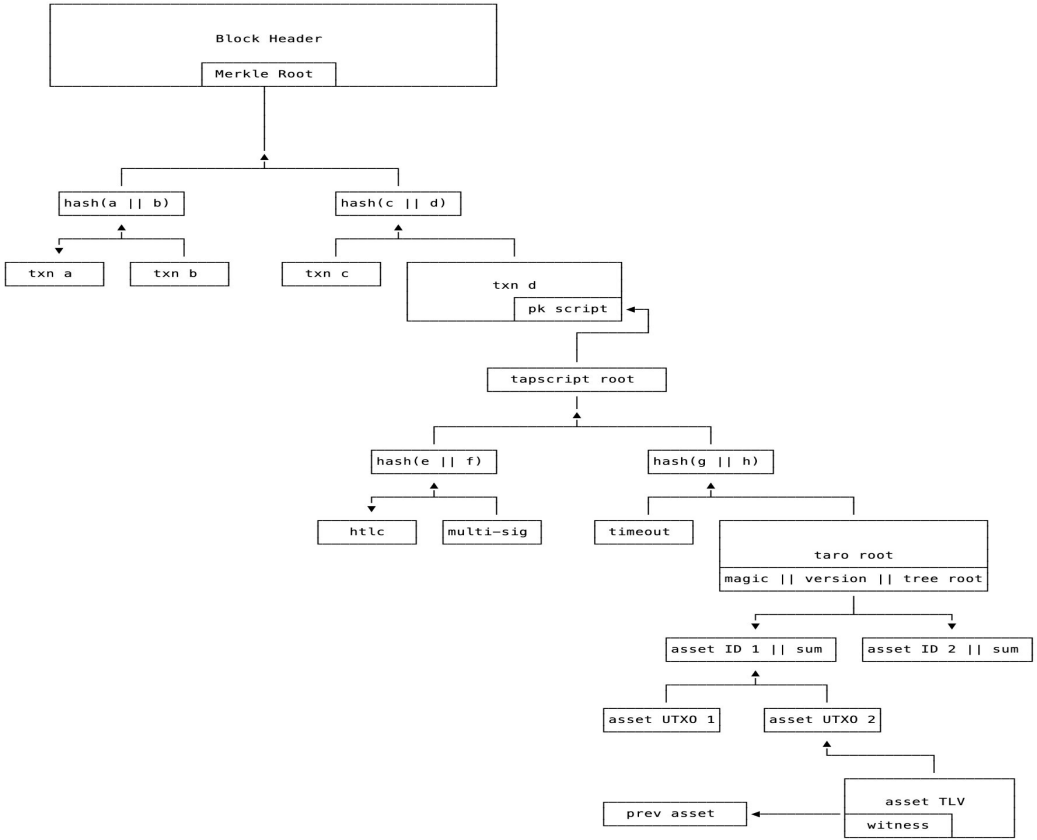
```
// The state of the headers chain and the UTXO set
struct State {
    // The state of the chain of block headers
    chain_state: ChainState,
    // The UTXO set represented as the roots of all trees in a Utreexo forest
    utreexo_roots: felt*,
}
```

```
// A summary of the current state of a block chain
//
struct ChainState {
    // The number of blocks in the longest chain
    block_height: felt,

    // The total amount of work in the longest chain
    total_work: felt,

    // The block_hash of the current chain tip
    best_block_hash: felt*,

    // The required difficulty for targets in this epoch
    // In Bitcoin Core this is the result of GetNextWorkRequired
    current_target: felt,

    // The start time used to recalibrate the current_target
    // after an epoch of about 2 weeks (exactly 2016 blocks)
    epoch_start_time: felt,

    // The timestamps of the 11 most recent blocks
    prev_timestamps: felt*,
}
```

Taro - Design Overview

- What is Taro?
  - Proof based bearer asset issuance+transfer system built on Bitcoin
    - An asset is a proof *explicit* (today) starting from a “genesis” asset, all rules were followed properly leading to final unspent asset output
      - f(prevAsset, assetWitness) -> newAsset
    - assetIDs generated via trick w/ BIP 34 (ensures coinbase uniqueness)
      - `assetID = sha256( genesisOutpoint || sha256(tag) || sha256(meta) || outputIndex || assetType )`
      - `assetType = Normal | Collectible`
    - Light client friendly! Normal assets can be sent using LN!
  - Leverages taproot, w/o off-chain data Taro transactions indistinguishable for normal transactions! (censorship resistance++)
  - Initial VM: Tapscript VM
  - Assets to committed in Merkle-Sum Sparse Merkle Tree rooted in tapscript leaf
    - Enables easy supply verification (“run the numbers”)
    - Application layer composition via swaps (exclusion proof for collectible, etc)

Taro - A Series of Nested Merkle Trees



Taro - Asset Proofs Today

```
taro % tarocli proofs decode --proof_file habibtaro_usd_jan2.json
{
    "proof_index": "0",
    "asset": {
        "version": 0,
        "asset_genesis": {
            "genesis_point": "3dea4202dc504302a68fa598ed6ade31f83fdc3a58b095f4d6259587fa48dea0:1",
            "name": "habibtaro_usd_jan_noem23",
            "meta": "68616269627461726f2e636f6d5f6a616e5f6e6f656d3233",
            "asset_id": "8074f97dc3f2a5c3375ddaccf66ffc0bd1fbdd0b8261ee536a1bc9653c0d9f0b",
            "output_index": 0,
            "genesis_bootstrap_info": "a0de48fa879525d6f495b0583adc3ff831de6aed98a58fa6024350dc0242ea3d0000000
            "version": 0
        },
        "asset_type": "NORMAL",
        "amount": "21000000",
        "lock_time": 0,
        "relative_lock_time": 0,
        "script_version": 0,
        "script_key": "020c7b5c4abd3b7cbe48a1cb8193312233924f22cfdc07e9c64a7f867cca3b5a3e",
        "asset_group": {
            "raw_group_key": null,
            "tweaked_group_key": "0253c2de331021b68632d5cb76110d4a65e4436f7762e47b8af058641012694110",
            "asset_id_sig": "cb61c6c21047b288e116880cb6abf941c754127a113e081e14b65cf834f445325817c4be9cdd44703
        },
        "chain_anchor": {
            "anchor_tx": "02000000000101a0de48fa879525d6f495b0583adc3ff831de6aed98a58fa6024350dc0242ea3d010000
            "anchor_txid": "986c8f9263aa5a0d1d6733019bc8660216b3e6a3380c38b73e39811881c677bd",
            "anchor_block_hash": "000000000004479a4e6a788f3994069b2c15b3b61b00673fd8a099323f4f6e78",
            "anchor_outpoint": "986c8f9263aa5a0d1d6733019bc8660216b3e6a3380c38b73e39811881c677bd:0",
            "internal_key": null
        }
    },
    "tx_merkle_proof": "06912174fe872bb334e87038d02313e51f85e76442f79b73b04e0c10b3330c84c2b2dc73a43a0cadf17126
    "inclusion_proof": "00040000000001210285127fca8f36e75b0508e6a6a561ab5db97003523f227596bc4f7be436a9b7a402c4
    "exclusion_proofs": [
        "00040000000012102c17d96c48980a0f48932ac66819e6ddc86cce68edaae0179d66d94326ce130160303020101"
    ]
}
```

Taro - Scalability Challenges

- Proof size growth over time
  - Proof size dominated by merkle tree inclusion proofs (3 layers of trees!)
  - Collectibles grow O(N), where N is the number of transfers
  - Normal assets can have inputs+outputs
    - Allows for normal BItcoin UTXO model, input fan in, output fan out, etc
    - Super linear growth based on branching factor deeper in history
    - Possible to optimize proofs after the fact to eliminate redundancy
      - For a given asset the transaction graph of any UTXO has a common ancestor
  - Ultimately bounded by growth of the Bitcoin blockchain itself
    - Taro transaction graph is a subset of the Bitcoin transaction graph
- Probabilistic verification 😎
  - Assemble state transitions into append-only merkle tree
  - Tree root commits to entire history of asset UTXO
  - Leverage fiat-shamir tricks to do challenge game of randomly select leafs
  - Verify enough until convinced of soundness
  - Can then recurse into the asset *inputs* themselves
    - `inputCommitment = merkleTree(input1, input2, etc)`

zk-Taro - Succinct Taro Proofs via Recursive STARKs

- Reduces down to the very same problem as STARK based Bitcoin blockchain sync
  - `f(prevAsset, witness) -> newAsset`
  - `f(prevTaroRoot, witness) -> newTaroRoot`
- Two layers of proofs, then recursion: 🔁
  - Inclusion:
    - "There exists a valid block header, with a merkle tree root that commits to a transaction, that itself commits to a valid Taro root within the pkscript of an output, and that Taro root contains an asset with attributes...."
  - State transition validity:
    - "The asset contains a valid witness based on public inputs ZZZ based on **rule set** YYY"
  - Recurse:
    - "Each of those inputs also has a valid inclusion + state transition validity proof...."
- End result is a single proof that itself *is* the asset
  - Each new transition feeds prior proof into recursion layer with new transition

zk-Taro - Architecture Design Sketches

- First-class STARK primitives
  - Can't modify sha256+ecdsa/schnorr Bitcoin layer
    - Optimize proof complexity via Cairo built-ins for the expensive operations
  - `taroRoot = sha256(magic, version, tree root)`
  - Version lets us re-design the Taro layer entirely
    - Can use more STARK friendly hash functions and crypto primitives
    - Removes all constraints of the Tapscript VM
    - Assets in the client-side validation layer gain general programmability
  - Can retain existing asset UTXO design, tree root commits to asset descriptions
    - Description can include normal type, amount, tag fields, etc
    - zk-Taro pk script:
      - `pkScript = h(programHash || publicInputs)`
      - Verifier doesn't need to know exact computation: privacy++
      - Public input similar to pkScript today: must generate sig w/ key, etc
    - zk-Taro witness:
      - STARK proof of prior state transition validity given below as starting w/ public inputs
    - Program output -> next set of public inputs (new pk for asset owner)

zk-Taro - Architecture Design Sketches

- Taro + zk-Taro Coexistence
  - Keep the current architecture basically as is
    - Still a young protocol, lots of learnings, optimizations to be had, etc
  - Apply STARK proofs to the *existing* proof format
    - Regular succinct inclusion proofs for the merkle/header/block layer (ZeroSync)
    - Prove STARK validity for the Tapscript VM based state transition of today
  - Requires full Bitcoin Script VM implementation in Cairo itself
    - "There exists a tapscript leaf in this commitment, where if you apply the witness data and normal validation rules, results in a final stack state where there's a single element that is 0x01"
      - `f(taprootKey, witness) -> (0x01, newTaprootKey)`
    - Today ZeroSync has implemented: ecdsa, sha256, ripemd160, etc
      - Missing Script VM itself, all soft forks since 2013, etc
    - Eventually wants Cairo built-ins for most expensive operations

zk-Taro - Architecture Design Sketches

- STARKs+Cairo as new Taro/Tapscript VM Version
  - assetTLV(..., ..., **scriptVersion**, **scriptKey**)
  - Today, version 0 means the Tapscript VM
    - scriptKey interpreted as normal taproot output key, run the same validation rules
  - Taproot added new type of script extensibility via tapscript version
    - `leaf = sha256(leafVersion || script)`
  - Leaf version allows soft forking in new features w/o modifying top level witness program version
  - Soft fork dry run:
    - New Taro VM version that signals awareness of Taro soft-fork for STARKs
    - Within this VM, define new leaf version that adds OP_STARKVERIFY
      - Assumes to be based on current/future Cairo architecture - verifier doesn't depend on the program being proved!
    - OP_STARKVERIFY
      - `<programHash> <publicInputs>`
      - Annex usage: `<starkProof>`
        - Annex used hypothetically to add new resourcing constraints to Bitcoin Script
        - `~SigOpCost -> f(traceLength)`

Future Directions

- Contribute to ZeroSync development
  - Most of the way there re initial Bitcoin validation
  - Still needs to catch up to latest set of soft forks
  - Understanding of tapscript layer needed to re-use for Taro inclusion proofs
- Investigation of Bitcoin accelerating Cairo built-ins
  - Bitcoin transaction serialization, double sha computation, secp256k1 operations, etc
- Implementing Taro primitives within Cairo
  - Asset TLV serialization
  - MS-SMT implementation
  - Verification of inclusion + exclusion proofs

We're hiring!

https://lightning.engineering/join-us

jobs@lightning.engineering